	FJER	CICIO 1										
N		LLAMADA RECURSIVA	RETURN									
4	F	factorial (3)	-									
	F											
3		factorial (2)	-									
2	F	factorial (1)	-									
1	F	factorial (0)	-					n = 0 → return 1				
0	V		1 1				n = 1 → 1 * factorial	n = 1 → 1 * factorial	return 1 * 1 = 1			
- 0	V	-	'			0 0 + fti-l	$n = 2 \rightarrow 2 * factorial$	(0)				
1	_	_	1x1= 1			(1) actorial	(1)	(1) actorial	(1) actorial	return 2 * 1 = 2		
					n = 3 → 3 * factorial	n = 3 → 3 * factorial	n = 3 → 3 * factorial					
2	-	-	2x1= 2	(2)	(2)	(2)	(2)	(2)	(2)	return 3 * 2 = 6	
				n = 4 \rightarrow 4 * factorial r	n = 4 → 4 * factorial	n = 4 \rightarrow 4 * factorial	n = 4 \rightarrow 4 * factorial	n = 4 \rightarrow 4 * factorial	n = 4 \rightarrow 4 * factorial	n = 4 \rightarrow 4 * factorial	n = 4 → 4 * factorial	
3	-	-	2x3= 6	(3)	3)	(3)	(3)	(3)	(3)	(3)	(3)	return
4	-	-	4x6= 24									
	EJERO	CICION 2										
LISTA	CONDICION (VACIA)	LLAMADA RECURSIVA	RETURN									
[a,b,c,d]	f	contar([b,c,d])	-									
[b,c,d]	f	contar([c,d])	-									
[c,d]	f	contar([d])	-									
[d]	f	contar([])	-									
0	v	- contar(jj)	0					lista = [] → return 0				
u u	*		, ,				lista = [d] → 1 +	lista = [d] → 1 +				
[d]	-	-	1+0=1				contar([])	contar([])	return 1 + 0 = 1			
						lista = [c,d] → 1 +	lista = [c,d] → 1 +	lista = [c,d] → 1 +	lista = [c,d] → 1 +			
[c,d]	-	-	1+1=2			contar([d])	contar([d])	contar([d])	contar([d])	return 1 + 1 = 2		
[b,c,d]			1+2=3		ista = [b,c,d] → 1 + contar([c,d])	lista = $[b,c,d] \rightarrow 1 +$ contar($[c,d]$)	lista = [b,c,d] → 1 + contar([c,d])	lista = $[b,c,d] \rightarrow 1 +$ contar($[c,d]$)	lista = $[b,c,d] \rightarrow 1 +$ contar($[c,d]$)	lista = $[b,c,d] \rightarrow 1 +$ contar($[c,d]$)	return 1 + 2 = 3	
[D,C,U]	-	<u> </u>	112-3	lista = [a,b,c,d] → 1 + li	W + 20		(C-) - D					
[a,b,c,d]	_	_	1+3=4	contar([b,c,d]) \rightarrow 1 \rightarrow 1 contar([b,c,d])	ontar([b,c,d])	contar([b,c,d])	contar([b,c,d])	contar([b,c,d])	contar([b,c,d])	contar([b,c,d])	contar([b,c,d])	return

	MATRIZ											
Entrada (fila/resto)	acc	Acción / Salida										
Entrada (ma/resto)	acc	Llama suma filas con										
[[1,2],[3,4]]	0	fila=[1,2]		# Caso base								
[[1,2],[2,1]]		Llama suma_fila con		<pre>defp suma_f</pre>	ilas([], aco	c), do: acc						
[1,2]	0	h=1										
[1,2]	0	Llama suma_fila con										
[2]	1	h=2		dofo cumo f	ilac/[fila	resto], acc	1) do					
	-			derp suma_i	1192([1119	restoj, acc	.) do					
	3	Caso base → retorna 3			is(resto, acc	c + suma_fila	a(T11a, 0))					
FF2 433	_	Llama suma_filas con		end								
[[3,4]]	3	fila=[3,4]										
FA 13	_	Llama suma_fila con		# Suma recu	ırsiva							
[3,4]	0	h=3			ila([], acc) do: acc						
		Llama suma_fila con		defp suma_f	iila([], acc	, do. acc	uma fila/+	oss (b)				
[4]	3	h=4			rra([u t]	, acc), do: s	suma_fila(t,	acc + n)				
	7	Caso base → retorna 7		end								
		Caso base final →										
	10	Salida total: 10										
	PERMI	JTACION		# Caso base								
	n	acum	Acción / Salida	<pre>defp permutar([</pre>], acum, n) do							
Lista (entrada)	0	0	Nivel 0 -> tomando :a	IO.puts("Nive	1 #{n} -> #{inspe	ect Enum.reverse(a						
	0	[:a]	Nivel 1 -> tomando :b	[Enum.reverse	(acum)]							
[:a, :b]			tomanuo .u	end								
	1	[.u]	Constant									
[:a, :b] [:b]	1		Caso base → imprime	# Caso recursiv								
[:a, :b]		[:b, :a]	[:a, :b]	# Caso recursiv defp permutar([do						
[:a, :b] [:b]	2	[:b, :a]	[:a, :b] Nivel 1 -> tomando :a	<pre>defp permutar([IO.puts("Nive</pre>	h t], acum, n) el #{n} -> tomando	o #{h}")						
[:a, :b] [:b]	1		[:a, :b] Nivel 1 -> tomando :a (segunda rama del for)	<pre>defp permutar([</pre>	h t], acum, n) l #{n} -> tomando est} <- separar([i	o #{h}") h t]) do						
[:a, :b] [:b]	1 2 1	[:b, :a]	[:a, :b] Nivel 1 → tomando :a (segunda rama del for) Caso base → imprime	<pre>defp permutar([</pre>	h t], acum, n) el #{n} -> tomando	o #{h}") h t]) do						
[:a, :b] [:b]	2	[:b, :a]	[:a, :b] Nivel 1 -> tomando :a (segunda rama del for)	<pre>defp permutar([IO.puts("Nive for {elem, re permutar(re end</pre>	h t], acum, n) el #{n} -> tomando est} <- separar([est, [elem acum	o #{h}") h t]) do						
[:a, :b] [:b]	1 2 1	[:b, :a]	[:a, :b] Nivel 1 → tomando :a (segunda rama del for) Caso base → imprime	<pre>defp permutar([</pre>	h t], acum, n) el #{n} -> tomando est} <- separar([est, [elem acum	o #{h}") h t]) do						

def calcular_exponente(_base, 0), do: 1										
def calcular_exponente(base, exponente) do										
base * calcular_exponente(base, exponente - 1)										
end										
					_					
				calcular_exponente(2, 0) exponente == 0						
				return 1						
			calcular_exponente(2, base = 2, exponente =	 calcular_exponente(2, 1) base = 2, exponente = 1 	calcular_exponente(2, 1) calcular 2 * 1					
				e(2,0) 2 * calcular_exponente(2,0						
		calcular_exponente(2, 2)	calcular_exponente(2,			calcular_exponents(7, 2)				
		base = 2, exponente = 2 2 * calcular_exponente(2, 1)	base = 2, exponente = 2 * calcular exponent	2 base = 2, exponente = 2 e(2, 1) 2 * calcular_exponente(2, 1)		calcular: 2 * 3 return 4				
	calcular_exponente(2, 3)	calcular_exponente(2, 3)	calcular_exponente(2,	3) calcular_exponente(2, 3)	calcular_exponente(2, 3)	calcular_exponente(2, 3)				
	base = 2, exponente = 3 2 * calcular exponente(2, 2)	base = 2, exponente = 3 2 * calcular_exponente(2, 2)	base = 2, exponente = 2 * calcular exponent	3 base = 2, exponente = 3 e(2, 2) 2 * calcular exponente(2, 2	base = 2, exponente = 3 2 * calcular exponente(2, 2)	base = 2, exponente = 3 2 * calcular exponente(2.1)	calcular: 2 = 2 2) semm.4			
	C Volkston Carte (Activities of the Carte (Carte)	To a company of the property of the	Tall Completed Medicine	EXCEPT CONTRACTOR BUILDING	A THE PERSONNEL PROPERTY OF THE PERSONNEL PR	C. Alexander Language and Co.	Comments.	•		
def buscar_elemento(_elem, []), do: false										
	true									
def buscar_elemento(_elem, []), do: false def buscar_elemento(elem, [elem _resto]), do:	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true			buca_tiemento(), (3.4)) siem = 3, head = 3						
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true									
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true			elem = 3, head = 3 setum frue buscar_elemento(3, [2, 3, 4])	Booker_thirteenin(3), [2,3]					
def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	true	elem = 3, h	head = 2, tail = [3, 4]	elem = 3, head = 3 return true buscar_elemento(3, [2, 3, 4]) elem = 3, head = 2, tail = [3, 4]	> buscar_elemento(3,[3,6					
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)		elem = 3, h buscar_ele	head = 2, tail = [3, 4] emento(3, [3, 4])	elem = 3, head = 3 return true buscar_elemento(3, [2, 3, 4]) elem = 3, head = 2, tail = [3, 4] buscar_elemento(3, [3, 4])	> buscar_elemento(3,[3,6]) return true	(1) - tme				
def buscar_elemento(elem, [elem _resto]), do: def buscar_elemento(elem, [_otro resto]) do buscar(elem, resto)	buscar_elements	elem = 3, h buscar_eler o(3, [1, 2, 3, 4]) buscar_eler = 1, tail = [2, 3, 4] elem = 3, h	head = 2, tail = [3, 4] emento(3, [3, 4]) emento(3, [1, 2, 3, 4]) head = 1, tail = [2, 3, 4]	elem = 3, head = 3 return true buscar_elemento(3, [2, 3, 4]) elem = 3, head = 2, tail = [3, 4]	buscar_elemento[3,[3,4] return true buscar_elemento(3, [1, 2 4] elem = 3, head = 1, tail =	(3, 4]) buscar_elem (2, 3, 4] buscar_elem	nenth(3, [1, 2, 3, 4]) nenth(3, [2, 3, 4]) = true			